Bhil$ C++

1. Affichage

```
#include <iostream> // Inclut la librairie d'entrée/sortie

using namespace std;
int main() {

    std::cout << "Bonjour, monde !" << std::endl;

    // std::endl ajoute un saut de ligne.
    return 0; // 0 indique que le programme s'est terminé sans erreur
}
```

2. 📝 Variables et Types de Données

En C++, le type définit l'espace mémoire alloué à la variable. La déclaration se fait toujours avant l'utilisation.

A. Déclaration et Types Primitifs

Type	Description	Exemple de Taille (peut varier)
int	Nombres entiers (positifs ou négatifs).	4 octets
double	Nombres décimaux (haute précision).	8 octets
char	Un seul caractère.	1 octet
bool	Valeurs booléennes (true ou false).	1 octet
std::string	Chaînes de caractères (nécessite #include <string>).	Variable

Exemple de Déclaration :

```
int age = 25;          // Déclare un entier
double poids = 75.5;   // Déclare un double
char initiale = 'A';   // Déclare un caractère (guillemets simples)
```

3. ⌨️ Entrées/Sorties Standard

On utilise la librairie `<iostream>` pour interagir avec l'utilisateur via la console.

- **Sortie (`cout`) :** Utilise l'opérateur d'insertion `<<`.
- **Entrée (`cin`) :** Utilise l'opérateur d'extraction `>>`.

```
#include <iostream>

using namespace std ;
int main() {
    int nombre;

    std::cout << "Entrez un nombre : ";
    std::cin >> nombre; // Lit l'entrée de l'utilisateur et la stocke dans 'nombre'

    std::cout << "Vous avez entré : " << nombre << std::endl;
    return 0;
}
```

B. Boucles (`for` et `while`)

La boucle `for` est la même que celle que nous avons vue en JS :

```
// Affiche les nombres de 0 à 4
for (int i = 0; i < 5; i++) {
    std::cout << i << " ";
}
// Résultat : 0 1 2 3 4
```

5. ⚙️ Fonctions

```
// Spécifie que la fonction retourne un entier (int)
int ajouter(int a, int b) {
    return a + b;
}

int main() {
    int resultat = ajouter(5, 8);
    std::cout << "Le résultat est : " << resultat << std::endl;
    return 0;
}
```

6. 🧠 Les Tableaux (Arrays)

Les tableaux sont statiques : leur taille doit être définie à l'avance et ne peut pas changer.

```
// Crée un tableau d'entiers de taille 4
int notes[4] = {12, 15, 8, 18};

// Accès : l'index commence toujours à 0
std::cout << notes[1] << std::endl; // Affiche 15

// Modification
notes[3] = 20; // La quatrième note est maintenant 20
```

🧮 1. Fonctions mathématiques (bibliothèque `<cmath>`)

Ces fonctions sont incontournables dès qu'on fait un peu de calcul :

Fonction	Description	Exemple
`abs(x)`	Valeur absolue d'un nombre	`abs(-5)` → `5`
`pow(a, b)`	Élève `a` à la puissance `b`	`pow(2, 3)` → `8`
`sqrt(x)`	Racine carrée	`sqrt(9)` → `3`
`ceil(x)`	Arrondit au **supérieur**	`ceil(4.3)` → `5`
`floor(x)`	Arrondit à l'**inférieur**	`floor(4.9)` → `4`
`round(x)`	Arrondit à l'entier le plus proche	`round(3.6)` → `4`
`fmod(a, b)`	Reste de la division flottante	`fmod(7.5, 2.0)` → `1.5`

🧵 2. Fonctions pour les chaînes de caractères (`<string>`)

Fonction	Description	Exemple
`s.length()` ou `s.size()`	Longueur de la chaîne	`"Hello".length()` → `5`
`s.substr(pos, n)`	Extrait une sous-chaîne	`"Bonjour".substr(0, 3)` → `"Bon"`
`s.find("mot")`	Cherche un mot dans une chaîne	`"Salut".find("lu")` → `2`
`s.replace(pos, n, "text")`	Remplace une partie de la chaîne	
`s.append("text")`	Ajoute du texte à la fin	`"Hi".append(" there")` → `"Hi there"`
`stoi(s)` / `stof(s)` / `stod(s)`	Convertit une chaîne en nombre	`stoi("123")` → `123`
`to_string(x)`	Convertit un nombre en chaîne	`to_string(42)` → `"42"`

📂 3. Fonctions d'entrée/sortie (`<iostream>`)

Fonction	Description	Exemple
`cin >> variable;`	Lit une donnée saisie par l'utilisateur	
`getline(cin, s);`	Lit une ligne complète (même avec espaces)	
`cout << "Texte";`	Affiche du texte à l'écran	
`endl`	Retour à la ligne et flush du buffer	`cout << "Hello" << endl;`

🧱 4. Fonctions sur les tableaux et vecteurs (`<algorithm>` et `<vector>`)

Fonction	Description	Exemple
`sort(begin(v), end(v))`	Trie un tableau ou vecteur	
`reverse(begin(v), end(v))`	Inverse l'ordre des éléments	
`max(a, b)` / `min(a, b)`	Donne le max ou min de deux valeurs	
`count(begin(v), end(v), x)`	Compte combien de fois `x` apparaît	
`find(begin(v), end(v), x)`	Cherche un élément	
`accumulate(begin(v), end(v), 0)` *(<numeric>)*	Fait la somme de tous les éléments	

📁 5. Fonctions de gestion de fichiers (`<fstream>`)

Fonction	Description	Exemple
`ofstream fichier("data.txt");`	Ouvre un fichier en écriture	
`fichier << "Texte";`	Écrit dans le fichier	
`ifstream fichier("data.txt");`	Ouvre un fichier en lecture	
`getline(fichier, ligne);`	Lit une ligne du fichier	

⚙️ 6. Fonctions utilitaires diverses

Fonction	Description	Exemple
`swap(a, b)`	Échange les valeurs de `a` et `b`	
`rand()` / `srand(time(0))`	Génère des nombres aléatoires	
`exit(0)`	Termine immédiatement le programme	
`system("cls")` ou `system("clear")`	Efface la console (Windows/Linux)	
`sizeof(var)`	Donne la taille en octets d'une variable	

💻 7. Programmation Orientée Objet (POO)

La POO est une approche de programmation qui organise le code autour de la notion d'**Objets**, qui regroupent à la fois des données (variables) et des fonctions (méthodes).

7.1. Les Classes (Classes)

Une **Classe** est le plan ou le moule qui sert à créer des Objets. Elle définit les propriétés (attributs) et les comportements (méthodes) que tous les objets de ce type auront.

Définition de la Classe

```
#include <iostream>
#include <string>
using namespace std;

class Utilisateur {
public:
    // Propriétés (Attributs)
    string nom;
    int age;

    // Comportement (Méthode)
    void saluer() {
        cout << "Bonjour, je suis " << nom << " et j'ai " << age << " ans." << endl;
    }
};
// SIMULATION CONSOLE (OUTPUT)
// Aucune sortie ici, c'est juste la définition du plan.
```

Modificateurs d'Accès Clés

En C++, il faut spécifier si les membres (propriétés et méthodes) sont publics ou privés.

- **`public`** : Les membres sont **accessibles** et modifiables de l'extérieur de la classe (ce que nous avons utilisé ci-dessus).
- **`private`** : Les membres sont **inaccessibles** de l'extérieur. Ils ne peuvent être modifiés que par les méthodes de la classe elle-même (c'est la règle d'**Encapsulation**). **C'est la pratique recommandée pour les attributs !**

7.2. Les Objets (Objects) : L'Instance

Un **Objet** est une **instance** concrète d'une classe. On déclare un objet de la même manière qu'une variable.

Création et Utilisation d'un Objet

```
// Dans la fonction main() :

int main() {
    // 1. Déclaration de deux objets de type Utilisateur
    Utilisateur u1;
    Utilisateur u2;

    // 2. Affectation des propriétés (utilisation de l'opérateur point '.')
    u1.nom = "Alice";
    u1.age = 30;

    u2.nom = "Bob";
    u2.age = 25;

    // 3. Appel des méthodes
    u1.saluer();
    u2.saluer();

    return 0;
}
```

```
// SIMULATION CONSOLE (OUTPUT)
// Bonjour, je suis Alice et j'ai 30 ans.
// Bonjour, je suis Bob et j'ai 25 ans.
```

7.3. Le Constructeur (Constructor)

Le **Constructeur** est une méthode spéciale de la classe. Elle est **appelée automatiquement** à la création de l'objet (l'instanciation). Son rôle principal est d'initialiser les attributs de l'objet.

- **Règle :** Le constructeur porte toujours le **même nom que la classe** et n'a **pas de type de retour** (même pas `void`).

Classe avec Constructeur

```
class Produit {
private:
    // Attributs privés (Encapsulation)
    string nom_produit;
    double prix;

public:
    // Constructeur avec paramètres pour initialiser l'objet
    Produit(string n, double p) {
        nom_produit = n;
        prix = p;
    }

    // Méthode pour afficher les détails
    void afficherDetails() {
        cout << "Produit : " << nom_produit << " | Prix : " << prix << " euros." << endl;
    }
};
```

Instanciation avec Constructeur

```
int main() {
    // L'objet est initialisé EN MÊME TEMPS que sa création
```

```
    Produit p1("Ordinateur", 1200.50);


    p1.afficherDetails();


    return 0;
}


// SIMULATION CONSOLE (OUTPUT)
// Produit : Ordinateur | Prix : 1200.50 euros.
```

7.4. **L'Encapsulation et les Accesseurs (Getters/Setters)**

Pour respecter la POO, on met les attributs en **`private`**. Pour permettre l'accès (lecture) ou la modification (écriture) contrôlée de ces attributs depuis l'extérieur, on utilise des méthodes publiques spéciales :

- **`Getter` (Accesseur) :** Permet de **lire** la valeur d'un attribut privé.
- **`Setter` (Mutateur) :** Permet de **modifier** la valeur d'un attribut privé, souvent avec une vérification de validité.

```
class CompteBancaire {
private:
    double solde;
public:
    // Constructeur
    CompteBancaire(double s) : solde(s) {} // Syntaxe d'initialisation concise


    // GETTER (Lecture)
    double getSolde() {
        return solde;
    }


    // SETTER (Écriture avec contrôle)
    void deposer(double montant) {
        if (montant > 0) {
            solde += montant;
```

```
            cout << "Dépôt réussi." << endl;
        } else {
            cout << "Montant invalide." << endl;
        }
    }
};




int main() {
    CompteBancaire monCompte(100.0);

    // On ne peut pas modifier monCompte.solde directement !

    // Lecture via le Getter
    cout << "Solde actuel : " << monCompte.getSolde() << endl;

    // Modification via le Setter (Dépôt)
    monCompte.deposer(50.0);

    cout << "Nouveau solde : " << monCompte.getSolde() << endl;
    return 0;
}

// SIMULATION CONSOLE (OUTPUT)
// Solde actuel : 100
// Dépôt réussi.
// Nouveau solde : 150
```